Unlink <modname> Usage : Unlinks module(s) from memory @WCREATE Syntax: Wcreate [opt] or /wX [-s=type] xpos ypos xsiz ysiz fcol bcol [bord] Usage : Initialize and create windows Opts : -? = display help -z = read command lines from stdin -s=type = set screen type for a window on a new screen @XMODE Syntax: Xmode <devname> [params] Usage : Displays or changes theparameters of an SCF type device @COCOPR Syntax: cocopr [<opts>] {<path> [<opts>]} Function: display file in specified format gets defaults from /dd/sys/env.file Options : -c set columns per page -f use form feed for trailer -h=num set number of lines after header -l=num set line length -m=num set left margin -n=num set starting line number and incr -o truncate lines longer than lnlen -p=num set number of lines per page -t=num number of lines in trailer -u do not use title

AUSTRALIAN OS9 NEWSLETTER

EDITOR	**Gordon Bentzen**	**(07) 344-3881**
SUB-EDITOR	**Bob Devries**	**(07) 372-7816**
TREASURER	**Don Berrie**	**(07) 375-1284**
LIBRARIAN	**Jean-Pierre Jacquet**	**(07) 372-4675**
	Fax Messages	**(07) 372-8325**
SUPPORT	**Brisbane OS9 Users Group**	

-u=title use specfied title -x=num set starting page number -z[=path] read file names from stdin or <path> if given @CONTROL Syntax: control [-e] Usage : Control Panel to set palettes, mouse and keyboard parameters and monitor type for Multi-Vue. Selectable from desk utilities menu as the Control Panel. Opts : -e = execute the environment file @GCLOCK Syntax: gclock Usage : Alarm clock utility for Multi-Vue. Selectable from desk utilities menu as Clock. @GCALC Syntax: gcalc Usage : Graphics calculator utility for Multi-Vue. Selectable form desk utilities menu as Calculator. @GCAL Syntax: gcal Usage : Calendar/Memo book utility for Multi-Vue. Selectable as Calendar from the desk utilities menu. @GPRINT

ADDRESSES

Editorial Material:
Gordon Bentzen
8 Odin Street
SUNNYBANK Qld 4109

Library Requests:
Jean-Pierre Jacquet
27 Hampton Street
DURACK Qld 4077

CONTENTS

Editorial ..... Page 2
Patches to Kernel ... Page 3
C Tutorial ..... Page 3
More Patches ..... Page 6
Murphy's Law ..... Page 7
Renewal Form ..... Page 10

Volume 6 **September 1992** **Number 8**

AUSTRALIAN OS9 NEWSLETTER
Newsletter of the National OS9 User Group
Volume 6 Number 8

EDITOR : Gordon Bentzen
SUBEDITOR : Bob Devries
TREASURER : Don Berrie
LIBRARIAN : Jean-Pierre Jacquet
SUPPORT : Brisbane OS9 Level 2 Users Group.

The National OS9 Usergroup Newsletter is no longer alone. We just have received a copy of the MOTD, the newsletter of the US OS9 Users Group. And a fine publication it is too. International subscriptions are available. You can send your subscriptions to:

The OS9 Users Group
PO Box 434
FARMINGTON
UTAH 84025 USA

Another person who is no longer alone is one of our members, Andrew Donaldson, the first person in Australia to get an MM/1. We now have another happy Australian MM/1 owner. None other than our own Don Berrie. So I guess that we can expect some comments and further information about the OSK systems from those users.

Don has promised to write a detailed article on his new system for a future newsletter, and I am looking forward to that. He will include tips on how to setup the boards in a case, operation of serial and parallel ports, installation of software, and other hardware. Not for the faint hearted I gather.

I guess that the news of the Australian CoCo Fest (if that's what we should call it) is big by now. Especially with those of our members who live in the Southern States. There is a good possibility that arrangements will be made to allow the National User Group to be represented there, and indeed, Bob Devries, Don Berrie, Jean-Pierre Jacquet (maybe) and myself are hoping to attend. It will be good to try to get to know some of the people who are currently just names on paper.

We welcome a couple of new additions to our PD library. Bob Devries has, through his involvement with the OS9 Community Network, managed to obtain copies of the latest releases of Scribe (an offline BBS mail processing system) and Stream (a fast hard disk backup system). It certainly is an improvement to be able to backup a hard drive and only take about 6 minutes to fill an 80 track (720K) disk!! These files are available from our librarian, Jean-Pierre. The usual copying charges will apply. Details of his address and phone number are on the front cover.

Speaking of the front cover, you will have noticed that, to celebrate the start of a new subscription year, we once again, have a new design for the front sheet. Don Berrie has once again produced a new the front cover. (Unfortunately, he didn't do it on his MM/1 though - maybe the next one ...)

One other change that will be obvious is the addition of a contents list on the cover page. This change has been incorporated in response to a request from one of our members. See, all you have to do is ask!!!

Speaking of asking, its the time of the year (isn't it always) when we take the opportunity to ask you to help fill the pages of our Newsletter. Remember, this is supposed to be a community effort, and without the input from our readers, we sometimes scratch for editorial material. If you would care to contribute to the Newsletter, you are more than welcome. No matter the level of the contribution, even questions, technical or software related, are welcome. (We also, at this time, would like to thank those who HAVE contributed in the past, and urge them to continue to do so.)

As this Newsletter is the first one of a new subscription year, and if you are one of those who have not yet renewed, you are receiving this issue on a complimentary basis, and this will be the LAST issue that you will receive. (Unless you resubscribe). If you do not intend resubscribing, we would like to thank you for your support in the past, and wish you the best of luck in your future computing endeavours. If you still wish to subscribe, then don't miss this opportunity!!

Cheers - Gordon

oooooooooo0000000000ooooooooo

A Patch to a Patch

by Bob Devries

I have been running a 1 Meg CoCo 3 for some time now, and have really enjoyed it. To make it possible to use OS9 and be able to use all that extra memory, a patch needs to be done to the 'OS9P1' part of the system. This file lives in the kernel, which resides on track 34 of the boot disk if you are using a floppy boot. The required patch is supplied with the package with the 1 Meg upgrade. The other thing the patch allows is use of OSK type filename, that is files starting with numbers, hyphens etc.

Well, word has come from USA that the patch has a bug, and needs to be patched. There seems to be a problem with 'data blocks' being allocated that were already being used by programmes. Some of the programmes that were affected were Stream (hard drive backup programme), and MVCanvas.

Here is the document supplied with some additional comments.

IMPORTANT NOTES ON USING REPACK

If you are using repack with a modified version of the OS9 kernel, please read the following instructions.

Some versions of the OS9 kernel that have been patched to support 1Meg/2Meg memory and OSK/MS-DOS file name formats contain a bug that will cause your system to crash when running REPACK.

The bug is in the F$AllImg service call, in module OS9P1. The bug causes OS9 to treat previously allocated 8K memory blocks containing OS9 modules as if they were writable data memory when allocating additional memory to a process. Under certain conditions, the process corrupts its own executable code by overwriting it with data.

The following patch to your modified OS9P1 module fixes the bug:

Offset	Old Value	New Value	
$083E	$85	$81	\These two lines added
$0840	$27	$26	/ omitted in original text
$09D7	$85	$81	
$09DB	$27	$26	

You must also fix the OS9P1 module CRC after making these changes.

Burke & Burke has included a utility called "kfix" on the File System Repack disk. This utility will correct the bug in OS9P1 and will automatically update the module CRC in memory. Note that you will have to use COBBLER or some other utility to save the fixed OS9P1 module to disk for a permanent correction. To use kfix, add the line:

```
kfix
```

to your STARTUP file.

--EOF--

As usual, this patch, in an archive called rpkfix.ar, will be available from Jean-Pierre Jacquet, our PD librarian.

Bob Devries

oooooooooo0000000000ooooooooo

A C Tutorial
Chapter 9 - Standard Input/Output
Part 2

WHICH METHOD IS BEST?

We have examined two methods of reading characters into a C program, and are faced with a choice of which one we should use. It really depends on the application because each method has advantages and disadvantages. Lets take a look at each. When using the first method, DOS is actually doing all of the work for us by storing the characters in an input buffer and signalling us when a full line has been entered. We could write a program that, for example, did a lot of calculations, then went to get some input. While we were doing the calculations, DOS would be accumulating a line of characters for us, and they would be there when we were ready for them. However, we could not read in single keystrokes because DOS would not report a buffer of characters to us until it recognized a carriage return. The second method, used in BETTERIN.C, allows us to get a single character, and act on it immediately. We do not have to wait until DOS decides we can have a line of characters. We cannot do anything else while we are waiting for a character because we are waiting for the input keystroke and tying up the entire machine. This

method is useful for highly interactive types of program interfaces. It is up to you as the programmer to decide which is best for your needs. I should mention at this point that there is also an "ungetch" function that works with the "getch" function. If you "getch" a character and find that you have gone one too far, you can "ungetch" it back to the input device. This simplifies some programs because you don't know that you don't want the character until you get it. You can only "ungetch" one character back to the input device, but that is sufficient to accomplish the task this function was designed for. It is difficult to demonstrate this function in a simple program so its use will be up to you to study when you need it. The discussion so far in this chapter, should be a good indication that, while the C programming language is very flexible, it does put a lot of responsibility on you as the programmer to keep many details in mind.

NOW TO READ IN SOME INTEGERS

Load and display the file named INTIN.C for an example of reading in some formatted data. The structure of this program is very similar to the last three except that we define an "int" type variable and loop until the variable somehow acquires the value of 100. Instead of reading in a character at a time, as we have in the last three files, we read in an entire integer value with one call using the function named "scanf". This function is very similar to the "printf" that you have been using for quite some time by now except that it is used for input instead of output. Examine the line with the "scanf" and you will notice that it does not ask for the variable "valin" directly, but gives the address of the variable since it expects to have a value returned from the function. Recall that a function must have the address of a variable in order to return the value to the calling program. Failing to supply a pointer in the "scanf" function is probably the most common problem encountered in using this function. The function "scanf" scans the input line until it finds the first data field. It ignores leading blanks and in this case, it reads integer characters until it finds a blank or an invalid decimal character, at which time it stops reading and returns the value. Remembering our discussion above about the way the DOS input buffer works, it should be clear that nothing is actually acted on until a complete line is entered and it is terminated by a carriage return. At this time, the buffer is input, and our program will search across the line reading all integer values it can find until the line is completely scanned. This is because we are in a loop and we tell it to find a value, print it, find another, print it, etc. If you enter several values on one line, it will read each one in succession and display the values. Entering the value of 100 will cause the program to terminate, and entering the value 100 with other values following, will cause termination before the following values are considered.

IT MAKES WRONG ANSWERS SOMETIMES

If you enter a number up to and including 32767, it will display correctly, but if you enter a larger number, it will appear to make an error. For example, if you enter the value 32768, it will display the value of -32768, entering the value 65536 will display as a zero. These are not errors but are caused by the way an integer is defined. The most significant bit of the 16 bit pattern available for the integer variable is the sign bit, so there are only 15 bits left for the value. The variable can therefore only have the values from -32768 to 32767, any other values are outside the range of integer variables. This is up to you to take care of in your programs. It is another example of the increased responsibility you must assume using C rather than a higher level language such as Pascal, Modula-2, etc. The above paragraph is true for most MS-DOS C compilers. There is a very small possibility that your compiler uses an integer value other than 16 bits. If that is the case, the same principles will be true but at different limits than those given above. Compile and run this program, entering several numbers on a line to see the results, and with varying numbers of blanks between the numbers. Try entering numbers that are too big to see what happens, and finally enter some invalid characters to see what the system does with nondecimal characters.

CHARACTER STRING INPUT

Load and display the file named STRINGIN.C for an example of reading a string variable. This program is identical to the last one except that instead of an integer variable, we have defined a string variable with an upper limit of 24 characters (remember that a string variable must have a null character at the end). The variable in the "scanf" does not need an & because "big" is an array variable and by definition it is already a pointer. This program should require no additional explanation. Compile and run it to see if it works the way you expect. You probably got a surprise when you ran it because it separated your sentence into separate words. When used in the string mode of input, "scanf" reads characters into the string until it comes to either the end of a line or a blank character. Therefore, it reads a word, finds the blank following it, and displays the result. Since we are in a loop, this program continues to read words until it exhausts the DOS input buffer. We have written this program to stop whenever it finds a capital X in column 1, but since the sentence is split up into individual words, it will stop anytime a word begins with capital X. Try entering a 5 word sentence with a capital X as the first character in the third word. You should get the first three words displayed, and the last two simply ignored when

the program stops. Try entering more than 24 characters to see what the program does. It should generate an error, but that will be highly dependent on the system you are using. In an actual program, it is your responsibility to count characters and stop when the input buffer is full. You may be getting the feeling that a lot of responsibility is placed on you when writing in C. It is, but you also get a lot of flexibility in the bargain too.

INPUT/OUTPUT PROGRAMMING IN C

C was not designed to be used as a language for lots of input and output, but as a systems language where a lot of internal operations are required. You would do well to use another language for I/O intensive programming, but C could be used if you desire. The keyboard input is very flexible, allowing you to get at the data in a very low level way, but very little help is given you. It is therefore up to you to take care of all of the bookkeeping chores associated with your required I/O operations. This may seem like a real pain in the neck, but in any given program, you only need to define your input routines once and then use them as needed. Don't let this worry you. As you gain experience with C, you will easily handle your I/O requirements. One final point must be made about these I/O functions. It is perfectly permissible to intermix "scanf" and "getchar" functions during read operations. In the same manner, it is also fine to intermix the output functions, "printf" and "putchar".

IN MEMORY I/O

The next operation may seem a little strange at first, but you will probably see lots of uses for it as you gain experience. Load the file named INMEM.C and display it for another type of I/O, one that never accesses the outside world, but stays in the computer. In INMEM.C, we define a few variables, then assign some values to the ones named "numbers" for illustrative purposes and then use a "sprintf" function. The function acts just like a normal "printf" function except that instead of printing the line of output to a device, it prints the line of formatted output to a character string in memory. In this case the string goes to the string variable "line", because that is the string name we inserted as the first argument in the "sprintf" function. The spaces after the 2nd %d were put there to illustrate that the next function will search properly across the line. We print the resulting string and find that the output is identical to what it would have been by using a "printf" instead of the "sprintf" in the first place. You will see that when you compile and run the program shortly. Since the generated string is still in memory, we can now read it with the function "sscanf". We tell the function in its first argument that "line" is the string to use for its input, and the remaining parts of the line are exactly what we would use if we were going to use the "scanf" function and read data from outside the computer. Note that it is essential that we use pointers to the data because we want to return data from a function. Just to illustrate that there are many ways to declare a pointer several methods are used, but all are pointers. The first two simply declare the address of the elements of the array, while the last three use the fact that "result", without the accompanying subscript, is a pointer. Just to keep it interesting, the values are read back in reverse order. Finally the values are displayed on the monitor.

IS THAT REALLY USEFUL?

It seems sort of silly to read input data from within the computer but it does have a real purpose. It is possible to read data in using any of the standard functions and then do a format conversion in memory. You could read in a line of data, look at a few significant characters, then use these formatted input routines to reduce the line of data to internal representation. That would sure beat writing your own data formatting routines.

STANDARD ERROR OUTPUT

Sometimes it is desirable to redirect the output from the standard output device to a file. However, you may still want the error messages to go to the standard output device, in our case the monitor. This next function allows you to do that. Load and display SPECIAL.C for an example of this new function. The program consists of a loop with two messages output, one to the standard output device and the other to the standard error device. The message to the standard error device is output with the function "fprintf" and includes the device name "stderr" as the first argument. Other than those two small changes, it is the same as our standard "printf" function. (You will see more of the "fprintf" function in the next chapter, but its operation fit in better as a part of this chapter.) Ignore the line with the "exit" for the moment, we will return to it. Compile and run this program, and you will find 12 lines of output on the monitor. To see the difference, run the program again with redirected output to a file named "STUFF" by entering the following line at the Dos prompt; A> special >stuff More information about I/O redirection can be found in your DOS manual. This time you will only get the 6 lines output to the standard error device, and if you look in your directory, you will find the file named "STUFF" containing the other 6 lines, those to the standard output device. You can use I/O redirection with any of the programs we have run so far, and as you may guess, you can also read from a file using I/O redirection but we will study a better way to read from a file in the next chapter.

WHAT ABOUT THE exit(4) STATEMENT?

Now to keep our promise about the exit(4) statement. Redisplay the file named SPECIAL.C on your monitor. The last statement simply exits the program and returns the value of 4 to DOS. Any number from 0 to 9 can be used in the parentheses for DOS communication. If you are operating in a BATCH file, this number can be tested with the "ERRORLEVEL" command. Most compilers that operate in several passes return a 1 with this mechanism to indicate that a fatal error has occurred and it would be a waste of time to go on to another pass resulting in even more errors. It is therefore wise to use a batch file for compiling programs and testing the returned value for errors. A check of the documentation for my COMPAQ, resulted in a minimal and confusing documentation of the "errorlevel" command, so a brief description of it is given in this file.

PROGRAMMING EXERCISE

1. Write a program to read in a character using a loop, and display the character in its normal "char" form. Also display it as a decimal number. Check for a dollar sign to use as the stop character. Use the "getch" form of input so it will print immediately. Hit some of the special keys, such as function keys, when you run the program for some surprises. You will get two inputs from the special keys, the first being a zero which is the indication to the system that a special key was hit.

oooooooooo0000000000ooooooooo

More patches...

For those of you who experience loss of data transmission while using a terminal program and CLEARing to a window with Multivue menus, the enclosed file provides a fix that should solve the problem. I remember that Kevin Darling mentioned these patches a long time ago on CIS, but I never got around to doing it for fear of screwing something up. The enclosed file was prepared by a user (Lee Veal I think) who figured it all out. Enjoy.

Hugo Bueno Delphi: MRGOOD Compuserve: 71211,3662
and coming soon UUCP: ...!bluehaus!hugo

I just put a patch in my WindInt module that eliminates a very bothersome (to me at least) occurence, when I'm using GShell. If you've used GShell, you've probably noticed that, when you use the <CLEAR> key to switch to a different window, the menu bar at the top of the screen does what I call a rollover. The line that used to contain the menu bar, now displays the program name and horizontal stripes, when the GShell window is not in the active window. To illustrate, open a Calendar from the Tandy Menu list. When another window is made the active window, the GShell window looses control of the keyboard and mouse. Watch closely, just before the Calendar screen is made the active window, you may see the process of the rollover start just before the GShell screen is replaced with the Calendar window.

What's the point? Well, the point is, that every time that rollover takes place, the routine that does the flip-flop seizes the system. In other words, it disallows any another process to run during the period of time that the rollover is being done. That's not a big deal most of the time, but it can be deadly, when one of those processes that gets "stiff-armed" is a communications program like SuperComm or XCom9, just to name two.

If you're downloading something using XMODEM with any communications program, and you flip away from the communications screen to the GShell window, then interrupts coming from the serial port will be lost. If interrupts are lost, so too, will be the data. Invariably, the Xmodem block that is being transmitted during the rollover will have to be re-transmitted, when the Xmodem routine detects the short block.

The following fix was applied to my system to eliminate the rollover entirely. Now, when I use the <CLEAR> key to switch from the communications screen to the GShell, no rollover occurs at all. And consequently no interrupts are lost, therefore no data are lost either.

This fix was given to me by Kevin Darling after a discussion of the above problem on CompuServe several weeks ago. I finally got off my big dead rear and applied the fix to my system at work. It worked perfectly.

The fix itself is extremely simple in its purpose, which is, if you, don't want the rollover to occur, then don't go to the routine that does the rollover. The application of the patch on the other hand may be a little more difficult. Not because the patch is complicated, but because several WindInt patches have been made public. Those patches may have caused the ML instructions that need to be changed to be in a different place than they were in the original version of WindInt.

Here goes anyway.

**** Warning ****

Do not apply this to your working copy of OS-9/Multi-Vue. Make a back up of your working copy and patch, then test with the backup copy.

Near $CA0 into WindInt should be something like

```
       25 03 17 03 91 9E A5
Change the    17 03 91
        to    12 12 12

   Then just past that around $CC5 you'll see
       25 03 17 07 1F 9E A5
Change the    17 07 1F
        to    12 12 12
```

According to Kevin, the numbers after the 17 in each case may not be exact, but the 2503 and the 9EA5 before and after each patch area should be the tipoff.

When I applied the fix to my WindInt module the displacements that Kevin had given ($CA0 and $CC5) were each a little lower than they needed to be. This was probably due to the different patches that we had in our respective WindInt modules.

I used DED to scan the OS9Boot (and the WindInt within it) for the 25 03 and the 9E A5 combos. I found them quickly and easily using DED.

I recommend that anyone thinking about applying this patch use DED to find and make the patch, then again to verify the module for the CRC update.

I've been using the patched modules for several days now with no ill-effects. And what's even better than that, is that now, I can switch through windows that have Multi-Vue menu bars without seeing the communications program suffer through lost data and lost interrupts. To me the obvious advantage of that is that, if I have a text editor up and running in one window, SuperComm running in another, and GShell in another, when I start a long download with SuperComm, and then I want to go over to text editor, I don't have to worry so much about lost interrupts, lost data, or block retransmits. Also, the screen switch seems to be faster. I say seems because you don't have to wait for the rollover.

For the technically-oriented the code that is replacing the 17 xx xx code is a series of three NOPs. A NOP is a mnemonic which stands for No OPeration. It holds a place in memory, allows the instruction processor to progress, but no data or registers are accessed, cleared, loaded, stored, or anything else during the "execution" of the NOP. A NOP instruction doesn't even effect the condition code.

The $17 op code that is being replaced by the $12s is a Long Branch to Subroutine (LBSR). The two bytes following the $17 is a 16-bit displacement that will be used to calculate the ultimate branch location from the end of the 3-byte LBSR instruction. The old LBSR instructions near $CA0 and $CC5 are being replaced by NOP instructions. Those LBSR instructions are branching to the routines that do the menu bar flip flop. Use this fix in good health. Lee

oooooooooo0000000000ooooooooo

MURPHY'S LAW AND OTHER OBSERVATIONS

Things could be worse; given enough time, they will be

Anyone who has had any contact at all with computers- building, repairing or programming them - knows Murphy's Laws. However, many people are unaware of the advent of machines and, later, computers.

The following series of observations have all been chosen because of their applicability to computerdom.

PREFACE:- We the willing, led by the unknowing, are doing the impossible for the ungrateful. We have done so much for so long with so little, we are now qualified to do anything with nothing.

MURPHY'S LAWS:-

1 - If anything can go wrong, it will.
2 - If there is a possibility of several things going wrong, the one that will cause the most damage will be the first one to go wrong.
3 - If anything just cannot go wrong, it will anyway.
4 - If you perceive that there are four possible ways in which something can go wrong, and circumvent these, then a fifth way, unprepared for, will promptly develop.
5 - Left to themselves, things tend to go from bad to worse.
6 - If everything seems to be going well, you have obviously overlooked something.
7 - Nature always sides with the hidden flaw.
8 - Mother nature is a bitch.

O'TOLLEL'S COMMENTARY ON MURPHY'S LAWS:- Murphy was an optimist.

GISBERG'S THEOREMS:-

1 - You can't win.
2 - You can't break even.
3 - You can't even quit the game.

FORTHY'S SECOND COLLARY TO MURPHY'S LAWS:- Just when you see the light at the end of the tunnel, the roof caves in.

WEILERS LAW:- Nothing is impossible for the man who doesn't have to do it himself.

THE LAWS OF COMPUTER PROGRAMMING:-
1 - Any given program, when running, is obsolete.
2 - Any given program costs more and takes longer each time it is run.
3 - If a program is usefull, it will have to be changed.
4 - If a program is useless, it will have to be documented.
5 - Any given program will expand to fill all available memory.
6 - The value of a program is inversely proportional to the weight of it's output.
7 - Program complexity grows until it exceeds the capability of the programmer who must maintain it.

PIERCE'S LAW:- In any computer system, the machine will always misinterpret, misconstrue, misprint or not evaluate any mathematics or subroutines or fail to print any output on at least the first run through.

COROLLARY TO PIERCE'S LAW:- When a compiler accepts a program without error on the first run, the program will not yield the desired output.

ADDITION TO MURPHY'S LAWS:- In nature, nothing is ever right. Therefore, if everything is going right.. something is wrong.

BROOKS LAW:- If a first you don't succeed, transform your data set!

GROSH'S LAW:- Computing power increases as the square of the cost.

GOLUB'S LAWS OF COMPUTERDON:-
1 - Fuzzy project objectives are used to avoid embarrassment of estimating the corresponding costs.
2 - A carelessly planned project takes three times longer to complete than expected, a carefully planned project takes only twice as long.
3 - The effort required to correct course geometrically with time.
4 - Project teams detest weekly progress reporting because it so vividly manifests their lack of progress.

OSBORN'S LAW:- Variables won't, constants aren't.

GIB'S LAWS OF UNRELIABILITY:-
1 - Computers are unreliable, but humans are even more unreliable.
2 - Any system that depends upon human reliability is unreliable.
3 - Undetectable errors are infinite in variety, in contrast to detectable errors, which by definition are limited.
4 - Investment in reliability will increase until it exceeds the probable cost of errors, or until someone insists on getting some useful work done.

LUBARSKY'S LAW OF CYBERNECTI ENTOMOLOGY:- There's always one more bug.

TROUTMAN'S POSTULATE:- Profanity is the one language understood by all programmers.

WEINBERG'S SECOND LAW:- If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.

GUMPERSON'S LAW:- The probability of anything happening is in inverse ratio to its desirability.

GUMMIDGE'S LAW:- The amount of expertise varies in inverse ratio to the number of statements understood by the general public.

ZYMURGY'S FIRST LAW OF EVOLVING SYSTEM DYNAMICS:- Once you open a can of worms, the only way to recan them is to use a larger can (old worms never die, they just worm their way into larger cans).

HARVARD'S LAW AS APPLIED TO COMPUTERS:- Under the most rigorously controlled conditions of pressure, temperature, volume, humidity and other variables, the computer will do as it damn well pleases.

SATTERING'S LAW:- It works better if you plug it in.

JENKINSON'S LAW:- It won't work.

HORNER'S FIVE-THUMB POSTULATE:- Experience varies directly with equipment ruined.

CHEOP'S LAW:- Nothing ever gets built on schedule or within budget.

RULE OF ACCURACY:- When working toward the solution of a problem, it always helps if you know the answer.

ZYMUG"S SEVENTH EXCEPTION TO MURPHY'S LAWS:- When it rains, it pours.

PUDDER'S LAWS:-
1 - Anything that begins well ends badly.
2 - Anything that begins badly ends worse.

WESTHEIMER'S RULE:- To estimate the time it takes to do a task: Estimate the time you think it should take, multiply by two and change the unit of measure to the next highest unit. Thus, we allocate two days for a one hour task.

STOCKMAYER' THEOREM:- If it looks easy, it's tough. If it looks tough it's damn near impossible.

ATWOOD'S COROLLARY:- No books are lost by lending except those you particularly wanted to keep.

JOHNSON'S THIRD LAW:- If you miss one issue of any magazine, it will be the next issue that contains the article, story or installment you were most anxious to read.

COROLLARY TO JOHNSON'S THIRD LAW:- All your friends either missed it, lost it or threw it out.

HARPER'S MAGAZINE LAW:- You never find the article until you replace it.

BROOK'S LAW:- Adding manpower to a late software project makes it later.

FINAGLE'S FOURTH LAW:- Once a job is fouled up, anything done to improve it will only make it worse.

FEATHERKILE'S RULE:- Whatever you did, that't what you planned.

FLAP'S LAW:- Any inanimate object, regardless of its position, configuration or purpose, may be expected to perform at any time in a totally unexpected manner for reasons that are either entirely obscure or elso completely mysterious.

MAINTENANCE RULES:-
1 - Put it back in the box and let someone elso fix it.
2 - Plug it in.
3 - Kick it.

oooooooooooOOOOOOOOOOooooooooooo

NATIONAL OS9 USERGROUP - AUSTRALIA

SUBSCRIPTION RENEWAL / APPLICATION

Subscription Renewal [__] New Subscription [__]

Surname: _____ First Name :_____ Title : ____

Street: _____

Suburb: _____ State :_____ Postcode : _____

Country:_____

Home Phone: _____ Business Phone: _____

Age Group (please tick) Under 18 [__] 18-25 [__] 26-35 [__]
36-45 [__] 46-55 [__] over 55 [__]

Do you run OS9 Level 1[__] OS9 Level 2[__] OSK[__] OS9000[__]

Type of Computer for OS9: _____ RAM _____K.

Diskette 5.25 Number [__] Tracks [___] Sides [__]

Diskette 3.5 Number [__] 720k [__] 1.4meg [__]

Hard Drive Meg [____] Controller Type _____

Printer Type/Model _____

Modem Type _____

Special Interests _____

_____

Can you contribute articles to this Newsletter ? _____

Date : ____/____/____ Signature :_____

Amount Enclosed: $ _____ ($18-00 will cover you for 12 months)
(A$25-00 Overseas)
(CHEQUES PAYABLE TO: **NATIONAL OS9 USERGROUP**)

Please Return Completed Form to :-

NATIONAL OS9 USERGROUP
c/o GORDON BENTZEN
8 ODIN STREET
SUNNYBANK QLD 4109

OR

NATIONAL OS9 USERGROUP
c/o J.P. JACQUET
27 HAMPTON STREET
DURACK QLD 4077